Compiler Exploitation of Decimal Floating-Point Hardware

Ian McIntosh, Ivan Sham

IBM Toronto Lab

Why do we need Decimal Floating Point?

- Microsoft® Office Excel 2003

	A	B	C
1	Value	1 Decimal Place	20 Decimal Place
2	1.95 - 1.85	0.1	0.09999999999999990000
3	0.1	0.1	0.10000000000000000000

…Why do we need Decimal Floating Point?

```
public static double calculateTotal(double
  price,                                double
  taxRate)
  {
  return price * (1.00 + taxRate);
  }
. . .
System.out.println("Total: $" +
          calculateTotal(7.0, 0.015));
-------------------------------------
Output -> Total: $7.1049999999999995
```

Outline

- IEEE Decimal Floating Point (DFP)
- C/C++ and DFP
- Java and DFP

What is IEEE 754-2008 Decimal Floating Point?

Type Name	Size	Precision	Exponent Range
decimal32	32 bits 4 bytes	7 digits single	-101 to +90
decimal64	64 bits 8 bytes	16 digits double	-398 to +369
decimal128	128 bits 16 bytes	34 digits quad	-6176 to +6111

What is Decimal Floating Point?

CASCON [2008]

- **Values use base 10 digits**
  - Alternative to Binary Floating Point

Sign bit

Combination field – encodes the first two bits of the exponent and the leftmost digit (BCD)

Exponent continuation – encodes the remaining biased exponent bits

Digits continuation – encodes the remaining digits in DPD 3-digit block form

Why should we use DFP?

- **Pervasive**
  - Decimal arithmetic is almost universal outside computers
- **More accurate for decimal numbers**
  - Can represent “important” numbers exactly
- **Programming trend**
  - IEEE 754, IEEE 854, IEEE 754R, IEEE 754-2008

Why should we use DFP?

- **Easier to convert to/from strings**
  - Great for working with databases
- **Performance**
  - More on this later

Why avoid using DFP?

- It's new and different
- Not all languages include DFP
- Limited support by other vendors
- Software implementations can be slow
- Incompatible formats (DPD and BID)
- Current IBM hardware is in most cases slower than binary floating point (BFP)

DFP at IBM

- **Hardware**
  - POWER6 and Z10
    - Microcode in Z9
  - One DFP functional unit
    - Non-pipelined
- **Software**
  - XL C, XL C++, gcc, PL/I
  - IBM® Developer Kit for Java™ 6

C Example – Without DFP

```
double calculateTotal(double price,
                      double taxRate)
  {
  return price * (1.00 + taxRate);
  }
. . .
printf ("Total: $%19.16f\n",
        calculateTotal(7.0, 0.015));
-----------------------------------
Output -> Total: $7.1049999999999995
```

C Example – With DFP

```
_Decimal64 calculateTotal(_Decimal64 price,
                          _Decimal64 taxRate)
  {
  return price * (1.00dd + taxRate);
  }
. . .
printf ("Total: $%19.16Df\n",
         calculateTotal(7.0dd, 0.015dd));
-----------------------------------------
Output -> Total: $7.1050000000000000
```

C / C++ DFP

C / C++ Type Name C++ Class Name	Literal Suffix	C printf / scanf Format Modifier	Library Function Suffix
_Decimal32 decimal32	df	HD	d32
_Decimal64 decimal64	dd	D	d64
_Decimal128 decimal128	dl	DD	d128

C / C++ DFP – Approaches

C syntax	Easiest and most natural. On AIX can be compiled to either use POWER 6 DFP instructions or call decNumber library. On z/OS uses DFP instructions.
DFPAL library	Automatically adapts to either using DFP instructions or calling decNumber.
decNumber library	Very portable library.
decFloat library	Newer and often faster library.
decNumber++ library	C++ DFP class library.

C/C++ DFP Performance – Product and Sum

In a loop: a[i] += b[i] * c[i];

	noopt	-O2	-O3
C syntax using decNumber library	(Baseline)	1.26x faster than noopt	2x faster than noopt
C syntax using DFP instructions	27x faster than software	1.82x faster than noopt 39x faster than software	4.37x faster than noopt 59x faster than software

Measured by Tommy Wong, Toronto Lab

xlc for AIX version 9 on POWER 6

C/C++ DFP Performance – C telco Benchmark

DFPAL* calls using decNumber	(Baseline)
decNumber calls	1.92x faster
DFPAL* calls using DFP instructions	2.56x faster
C syntax using DFP instructions	4.4x faster

* DFPAL automatically adapts to either using DFP instructions or calling decNumber.

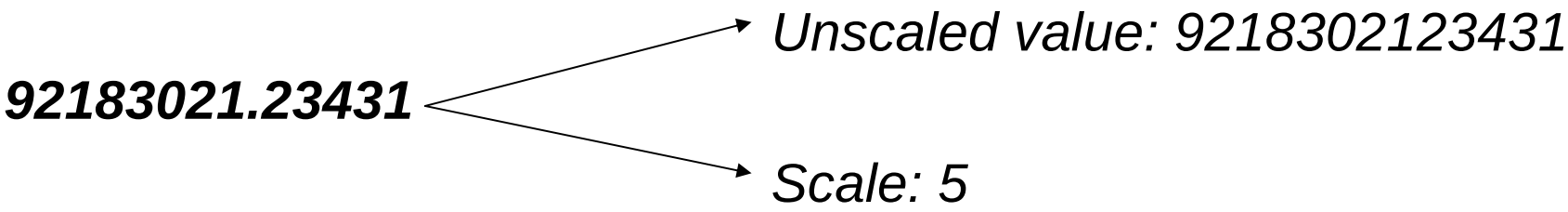
Measured by Tommy Tse, Beaverton
xlc for AIX version 9 on POWER 6 using -O2

Decimal Floating Point in Java

- IBM Developer Kit for Java 6
- 64 bit DFP via BigDecimal class library
- POWER 6 server or Z10 mainframe

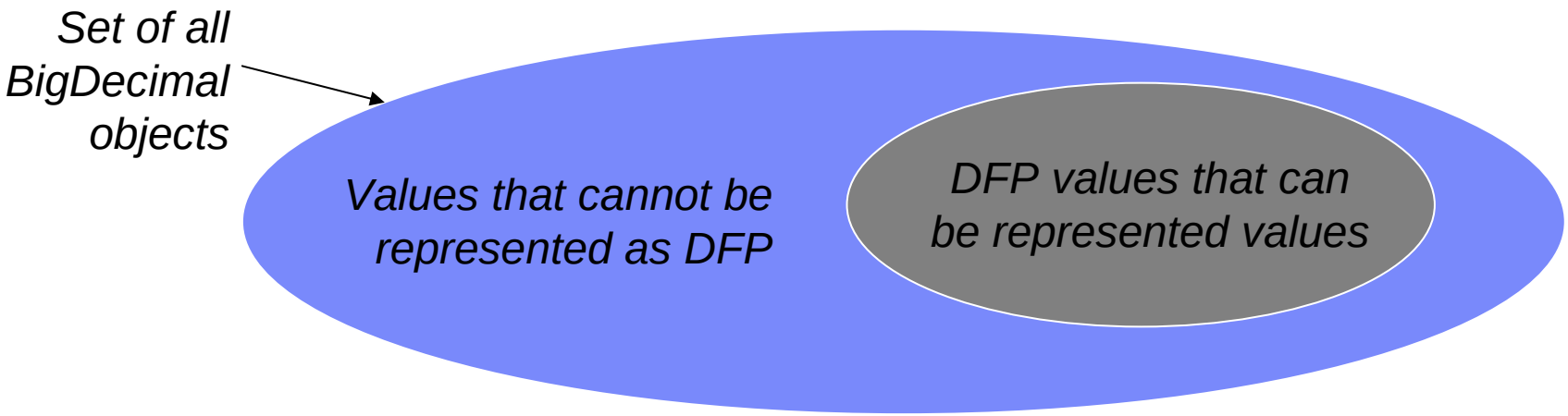
BigDecimal Class Library

- **arbitrary-precision signed decimal numbers**
  - an arbitrary precision integer *unscaled value*
  - 32-bit integer *scale*
- **Supports all basic arithmetic operations**
- **Complete control over precision and rounding behavior**



BigDecimal and DFP

- BigDecimal can represent arbitrary significance but 64-bit DFP restricted to 16 digits
- BigDecimal represents 32-bit exponent, 64-bit DFP restricted to 10 bits



BigDecimal Representation Problem

- **Want to:**
  - Use DFP representation
  - Avoid software re-try

```
BigDecimal a = new BigDecimal("9876543210123456",
                              MathContext.DECIMAL64);
BigDecimal b = new BigDecimal("1234567890987654",
                              MathContext.DECIMAL64);
```

Fits in 64 bit DFP

```
BigDecimal c = a.add(b);
```

Precision overflow

Hysteresis Mechanism

- **Choose best representation automatically**
  - Base on history of operations
- **Use counter and threshold**
  - Bias towards DFP representation
    - Division, string construction, unaligned addition
  - Bias towards software representation
    - Compare, integer constructions
- **BigDecimal constructors check counter**

JIT Compiler Optimization

- **Detects DFP hardware support**
  - Replaces checks in java code with constant
  - Disables hysteresis mechanism when no DFP

- **Inject DFP instructions**
  - Load operands from BigDecimal Objects
  - Set rounding mode (if necessary)
  - Perform DFP operation
  - Reset rounding mode (if necessary)
  - Check result validity
  - Store result into BigDecimal Object

Example – Java / BigDecimal

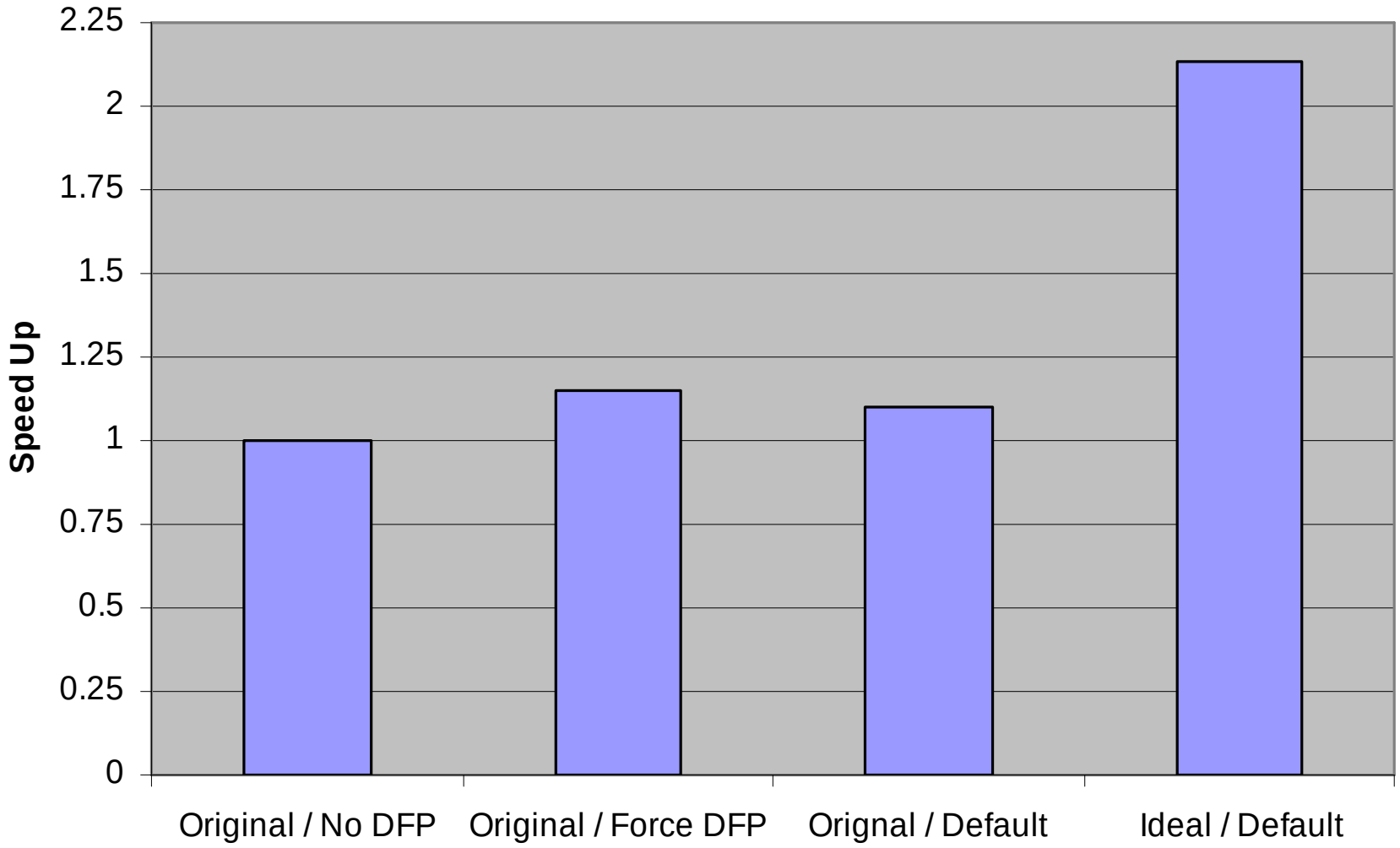
```
public static BigDecimal calculateTotal(
      BigDecimal price, BigDecimal taxRate)
  {
  return price.multiply(taxRate.add(BigDecimal.ONE));
  }
. . .

System.out.println("Total: $" + calculateTotal(
  new BigDecimal(“7.00”), new BigDecimal(“0.015”));

----------------------------------------
Output -> Total: $7.1050
```

Microbenchmark results

	HW DFP Speed up
Unaligned Addition	**5.05x**
Aligned Multiplication	**3.03x**
Aligned Division	**2.23x**
Half Even Rounding	**1.45x**
String based construction	**2.08x**

zLinux on Z10 using Java 6 SR2

Performance Improvement - Telco



z/OS on Z10 using Java6 SR1

Summary

- **Use DFP**
  - Control over precision and rounding behaviour
  - Accuracy for decimal numbers
  - Programming trend
- **High performance for suitable workloads**
  - DFP hardware can greatly improve performance
  - 4x (2x) speedup was measured on C (Java) for Telco

Thank you!

IBM Toronto Software Lab

Ian McIntosh ianm@ca.ibm.com

Ivan Sham ivansham@ca.ibm.com

Resources

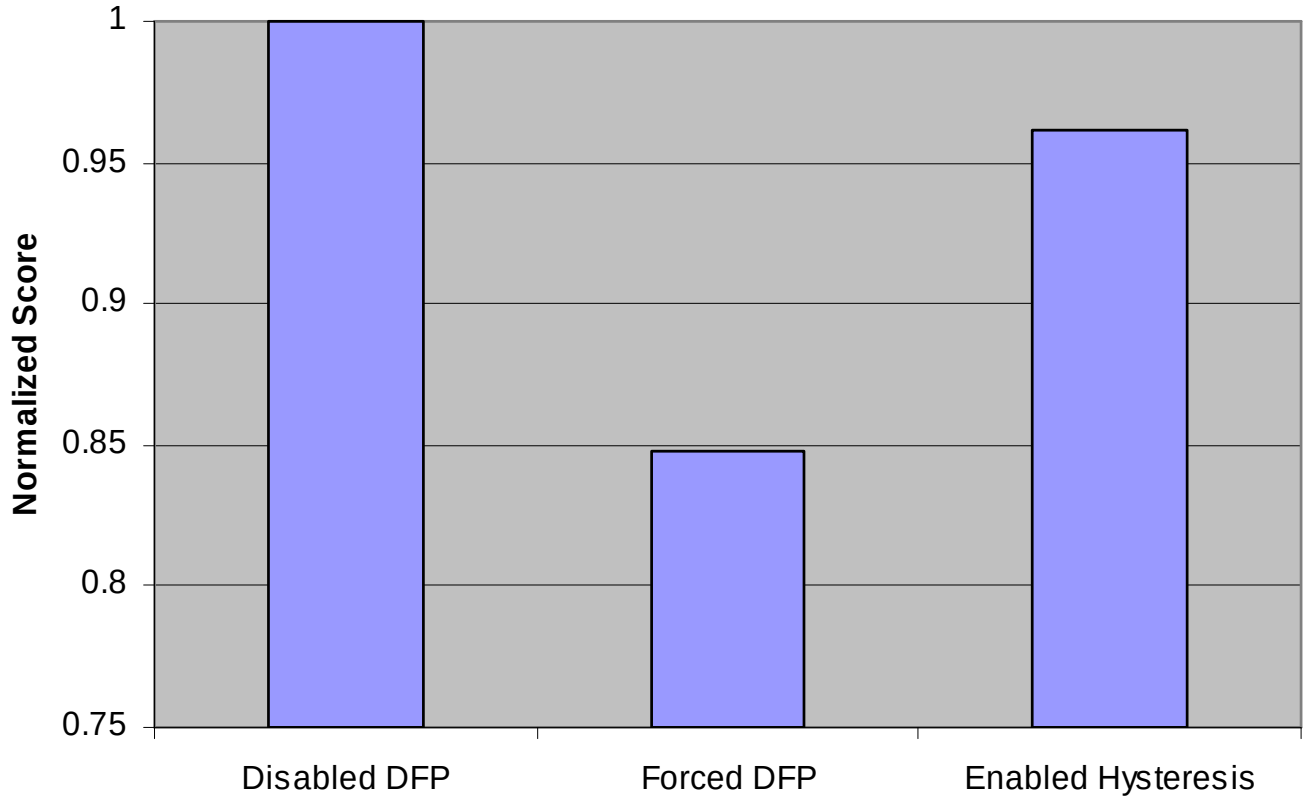
- **General Decimal Arithematic**
  - http://www2.hursley.ibm.com/decimal/
- **Decimal floating-point in Java 6: Best practices**
  - https://www.304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/whitepaper/power/java6_sdk/best_practice

Java command line options

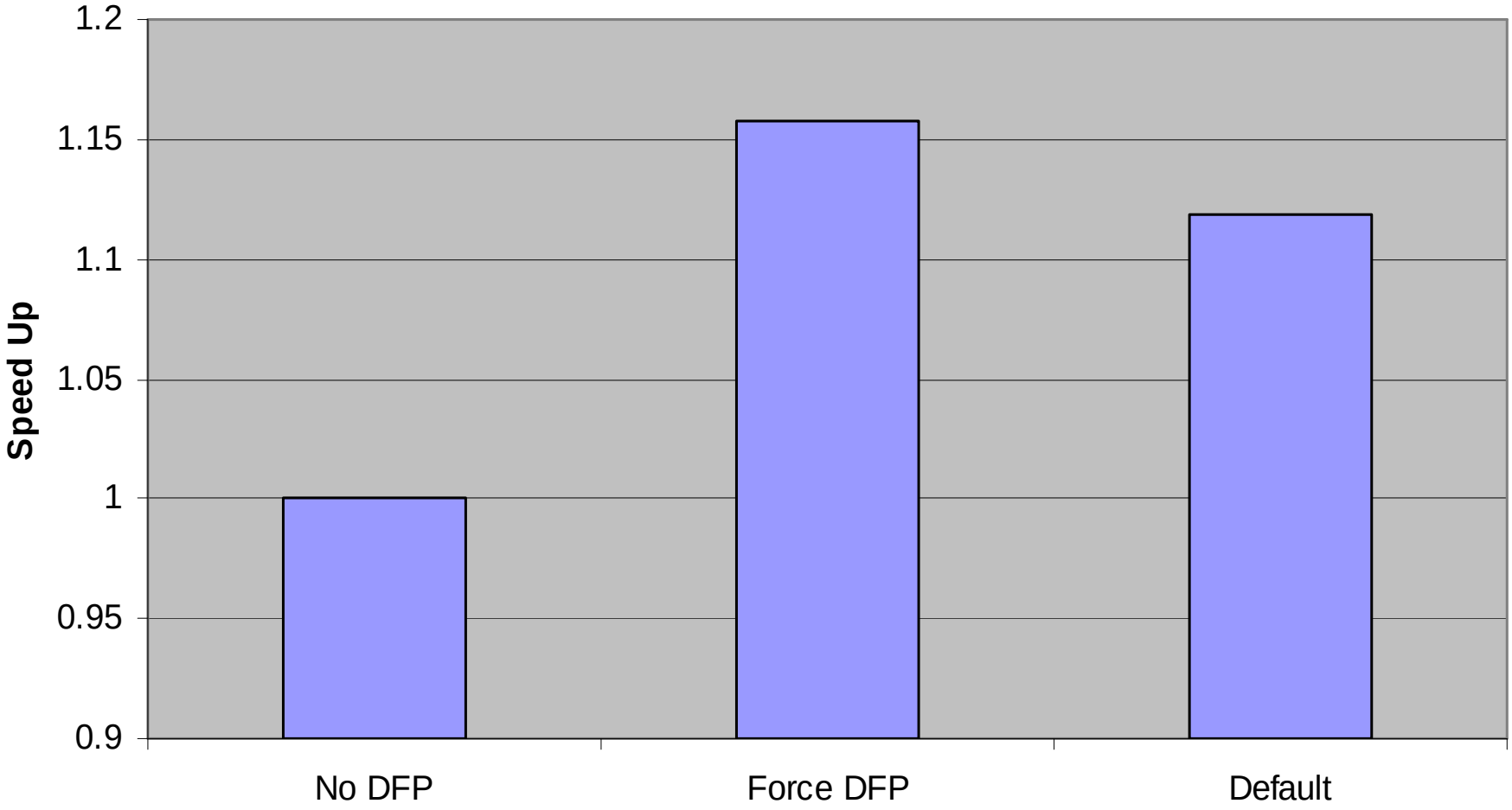
- **-Xdfpbd**
  - Disables the hysteresis mechanism
- **-Xnodfpbd**
  - Disable DFP support and hysteresis mechanism

Hysteresis Mechanism Performance

- **Multi-threaded transaction base benchmark**
  - Workload does not use MathContext64



Java Telco Performance on POWER6



AIX on POWER6 using Java 6